FCA in a Logical Programming Setting for Visualization-oriented Graph Compression

Lucas Bourneuf✉ and Jacques Nicolas✉

Université de Rennes 1 and INRIA centre de Rennes
Campus de Beaulieu, 35042 Rennes cedex, France.
{lucas.bourneuf,jacques.nicolas}@inria.fr

Abstract. Molecular biology produces and accumulates huge amounts of data that are generally integrated within graphs of molecules linked by various interactions. Exploring potentially interesting substructures (clusters, motifs) within such graphs requires proper abstraction and visualization methods. Most layout techniques (edge and nodes spatial organization) prove insufficient in this case. Royer et al. introduced in 2008 *Power graph* analysis, a dedicated program using classes of nodes with similar properties and classes of edges linking node classes to achieve a lossless graph compression. The contributions of this paper are twofold. First, we formulate and study this issue in the framework of Formal Concept Analysis. This leads to a generalized view of the initial problem offering new variants and solving approaches. Second, we state the FCA modeling problem in a logical setting, Answer Set programming, which provides a great flexibility for the specification of concept search spaces.

Keywords: graph compression, graph visualization, bioinformatics, ASP

1 Introduction: graph compression for graph visualization

Large graphs are a common entry of many application domains including information systems, program dependency graphs, social networks, and experimental data. We are particularly interested in molecular biology that accumulates huge amount of data, generally integrated within graphs of molecule interactions. Understanding the main structures present in such graphs is a source of knowledge that goes far beyond general statistics on the graph topology and can lead to the discovery of key organization schema reflecting disease determinants, regulation mechanisms or active domains. A common way of studying them uses visualization methods that focus on smart displays organizing spatially the edges and the nodes or using virtual nodes aggregating structural information [12, 18]. A more powerful approach consists in first summarizing the graph and then using this compressed representation for visualization or graph mining.

1.1 Graph Compression

Graph compression looks for possible node or edge aggregations, i.e. connections between clusters of nodes instead of connections between individual nodes.

Not surprisingly, it has been shown that graph readability increases with edge compression [9]. Among early works in this direction, Agarwal *et al.* [1] have shown that visibility graphs, a type of graph commonly used in computational geometry, can be represented compactly as a union of cliques and bicliques. The decomposition (partition of the edges) or the covering (multiple use of edges) of graphs into subgraphs belonging to a particular family have been the subject of many studies. Bounds on the size complexity of such coverings have been early established for the important particular case of complete bipartite subgraphs [7] but many interesting open combinatorial problems remain in this area [17]. From an algorithmic perspective, the generation of all maximal bicliques of a graph is related to and may be considered as an important subtask of the covering problem. Apart from algorithms developed in Formal Concept Analysis, applied mathematics have also worked on classes of graphs for which it is possible to find the set of all maximal bicliques in time polynomial in the size of the graph. For instance, it is possible to find linear time algorithms for the case of graphs of bounded arboricity [11] or for domino-free graphs [4]. For general graphs, the best one can hope is to get total polynomial algorithms, i.e., polynomial with respect to the size of the union (input + output). This has been proposed in [3], with an algorithm derived from the consensus method.

On the practical side, a nice visualization of compressed graphs introduces additional constraints on the choices of subgraphs that add a complexity level in the covering or decomposition problem. We have already mentioned that it is useful to allow both cliques and bicliques for more compact representations. introduced by Royer *et al.*, the *Power graph* analysis is a clustering and visualization method [23] that starts from this requirement and has been specifically designed to show these subgraphs typically arising in bioinformatics. Indeed, they have been associated to important structures in biological networks, particularly for protein interactions [2, 21, 12]. Bicliques also show interactions induced by specific protein domains in case of protein-protein interactions, or for multi-target drugs in case of drug/target/disease networks [8]. Furthermore, this approach has been used as an alternative approach to compare two biological networks by measuring the compression ratio of the compressed union of the graphs [22]. Due to the genericity of the subgraph motifs, *Power graph* analysis has been used for applications in other research fields like reliable community detection in social networks [26].

1.2 Power Graph

Given a graph $G = (V, E)$, a power graph is defined as a special graph $PG = (PV, PE)$ where the nodes PV are subsets of V and the edges PE are subsets of E. A power graph must fulfill the three following conditions:

subgraph condition Any pair of power nodes connected by a power edge represents a biclique in graph G. As a special case, a clique in G is represented by a single power node and a power edge looping on this node.

power node hierarchy condition Any two power nodes are either disjoint, or one is included in the other. From the point of view of classification, the sets of vertices clustered in power nodes form a hierarchy.

power edge decomposition condition power edges form a partition of the set of edges.

An example of graph compression is shown in Figure 1.

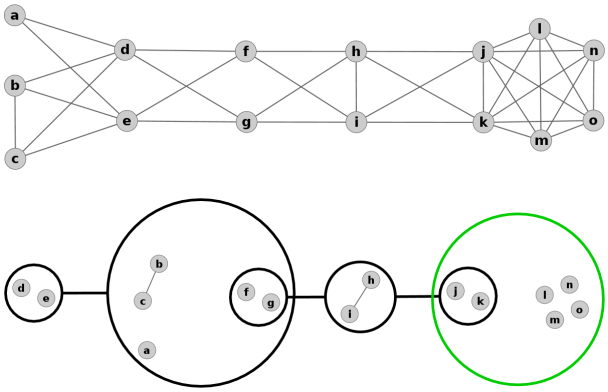

Fig. 1: A graph with 15 nodes labelled from a to o and 35 edges. A smart layout of this graph allows one to understand its underlying structure, something that becomes hard with a growing number of edges. The bottom compressed version of this graph has been produced by *Power graph* analysis and printed with Cytoscape [25], through a plug-in developed by Royer *et al.*. Power edges are shown as thick and black lines linking power nodes (thick circles, black for bicliques and green/grey for cliques). Some edges like (h, i) remain uncompressed. Concept $(\{h, i\}, \{f, g, j, k\})$, despite being a maximal biclique, can't be associated to a single power edge without breaking the power node hierarchy condition. Instead, two power edges are generated, corresponding to bicliques $(\{h, i\}, \{f, g\})$ and $(\{h, i\}, \{j, k\})$. The same way, the subgraph on the subset of vertices $\{d, e, f, g, h, i, j, k\}$ can't be covered by power edges $(\{f, g\}, \{d, e, h, i\})$ and $(\{h, i\}, \{f, g, j, k\})$ since it would break the power edge decomposition condition
.

The issue is to exhibit a power graph with a minimal number of power edges. It is not necessarily unique. It has been shown to be a NP-complete problem [10]. An algorithm and a software are described in [23]. It implements a two-phase approach, first processing the possible power nodes by a hierarchical clustering of the nodes using the Jaccard index on the sets of neighbours, and then building the power edges than can be drawn between any pair of power nodes following a greedy incremental approach, choosing at each step a maximal subgraph in the number of covered edges. The algorithm is very fast but remains heuristic and only computes an approximation of the minimal powergraphs. It also has been implemented with slight variations, for instance to work on directed graphs, or to enable overlapping power nodes for visualization of relations between non-disjoints sets [2], or to enable an edge to be used multiple times for a faster search for near-to-optimal compression [10].

The paper contributions are twofold. First, we formulate and study this issue in the framework of FCA. The goal is to be able to benefit from the advances in this domain to get a better view of the structure of the search space, suggesting variants and solving approaches and conversely to offer a playground for new studies in FCA. Second, we have stated the FCA modeling problem in a logical setting, Answer Set programming (ASP). The first proposition of ASP program for FCA seems due to CV Damásio and published only in [16], where the focus is on the developing expressive query languages for formal contexts. A more recent study extends the search for n-adic FCA [24] and is focused on the issue of filtering large concept spaces by checking additional membership queries. In our case, the particular task of graph compression needs heavier calculations and a specific code since the whole concept space need to be explored.

Our goal is to show that this high level language provides a great flexibility in the specification of concept search spaces and thus enables to easily explore the effect of new constraints or new properties on the lattice of concepts. We have implemented a modeling of *Power graph* compression as a formal concept search. We illustrate in the last section some results showing that the developed tool is already applicable to real biological applications.

2 Power Graph as a Formal Concept Search

This section first defines how the search space for *Power graph* compression can be formalized with FCA. The section ends on a proposition working on formal contexts, allowing one to split the search space of power graphs.

2.1 Motifs as Formal Concepts

Power graph compression leads to edge reduction through motif recognition: bicliques and cliques are abstracted in power nodes and power edges. A power node is a node that represents a non empty set of (power) nodes. Power edges link (power) nodes. A biclique is compressed in two power nodes, one for each set, linked by a power edge. Cliques are represented by a single power node with a reflexive power edge. Stars are particular case of bicliques where a power edge links a single node with a power node. These motifs can be ordered by edge coverage: a motif is greater than another if it covers more edges. For instance, a biclique of 2×4 nodes has an edge coverage of 8, which is larger than a clique of 4 nodes that covers 6 edges. This is the spirit of the Royer *et al.* heuristic.

A *graph context* is a formal context where objects and attributes are nodes in the graph and the binary relation represents edges. For a bipartite graph, the relation may be oriented and objects and attributes are distinct. In the general case, objects and attributes represent the same elements and the graph context is symmetric, comparable to an adjacency matrix. We use lower case for objects and upper case for attributes to distinguish them (see Table 1).

Building a power graph consists in iteratively looking for the largest motifs in the graph (cliques or bicliques) until all edges are covered. Largest bicliques

correspond to formal concepts in the graph context where objects and attributes are disjoint (e.g. $\{a, b\} \times \{g, h, i\}$ in Table 1). Largest cliques correspond to formal concepts where objects and attributes are equal. The other cases where objects and attributes partially overlap are not allowed.

Yet the motifs compressed by *Power graph* compression are not always formal concepts themselves, because, in order to respect the power node hierarchy and power edge decomposition conditions, maximal (bi)cliques are not always fully compressed, as indicated in the introduction and illustrated in Figure 1.

Let (A, B) and (C, D) be two formal concepts, the power node hierarchy condition is not fulfilled if two sets are partially overlapping, that is $X \cap Y \notin \{\emptyset, X, Y\}$ for $X \in \{A, B\}$ and $Y \in \{C, D\}$. This is illustrated in Table 1 for $X = B$ and $Y = C$, with $A = \{a, b\}$, $B = \{g, h, i\}$, $C = \{e, f, i\}$, and $D = \{c, d\}$. In such a case, one of the concept has to be covered by two power edges, using a simple set algebra: (C, D) can be covered by $D \times (C \setminus B)$ and $D \times (C \cap B)$).

Similarly, let (A, B) and (C, D) be two formal concepts, the power edge decomposition condition is not fulfilled if two pairs of sets have a common intersection. Such a situation commonly occurs when the concepts have crossed dependencies: $A \subset D$ and $C \subset B$. In Figure 1 for instance, with $A = \{f, g\}$, $B = \{d, e, h, i\}$, $C = \{h, i\}$, $D = \{f, g, j, k\}$ the same edges $(f, h), (g, h), (f, i), (g, i)$ are represented multiple times. In such a case, one of the concept has to be restricted: (C, D) can be covered by $(C, D \setminus A)$.

In summary, power graph compression may be seen as a contextual search of concepts in the graph. The effect of the two admissibility constraints required by power graphs are different on the concepts associated to power edges: the power edge decomposition condition does not increase the number of bicliques in the decomposition while the power node hierarchy condition has a more drastic effect by splitting concepts. Finally, it has to be noticed that this problem introduces an interesting covering problem in FCA: how to extract a cover of all edges by a minimum number of concepts.

	A	B	C	D	E	F	G	H	I
a							x	x	x
b							x	x	x
c					x	x			x
d					x	x			x
e			x	x					
f			x	x					
g	x	x							
h	x	x							
i	x	x	x	x					

Table 1: Illustration of the power node hierarchy condition on a graph context: Assume the grey cells mark an already selected motif (here, biclique $\{a, b\} \times \{g, h, i\}$). Although the concept $\{c, d\} \times \{e, f, i\}$ is valid as a standalone motif, it overlaps the previous one and these two motifs cannot occur together.

2.2 Heuristics Modeling

As already discussed, the Royer *et al.* heuristics consists of an iterative search of the largest motifs to compress. From the FCA point of view, this can be modeled by an iterative search of the largest concept and the suppression of covered edges at each step from the graph context until an empty context is reached. Note that in this way, we can possibly obtain a better decomposition solution than Power graph Analysis since the choice of the best concept is evaluated at each step in our case whereas all bicliques are ranked only once during the initial step in the other case. We give in section 3 the programs that implement this search.

The Dwyer *et al.* heuristics is much more complex and can reach an optimal compression. The idea is basically to first create all stars associated with each node, then iteratively merge two (power) nodes involving a maximal number of common neighbors. The approach seems to be compatible with Chein algorithm [6] but it is not clear that it fulfill all conditions of powergraphs and we have not further explored this track of research. Instead, we have designed a concept generation heuristic, the *HDF* method (for High Degree First). At each step, the HDF heuristics selects first nodes of highest degree, then the largest concept using one of these nodes. The degree is just one measure on the graph topology and could be replaced by any notion of *node degree of interest*. We have tried a variant, *K2HDF*, directly inspired from Dwyer *et al.* heuristics. In this variant, the selected nodes include nodes of highest degree and nodes with the highest number of neighbours in common with another node. HDF is quicker than the *greater-first* heuristics, to the expense of a slightly smaller edge reduction (see section 4). We have tested another variant, called Fuzzy HDF (FHDF). The idea is to directly use the degree of interest for scoring. FHDF tries to maximize the total degree of interest of nodes involved in the concept. It provides an upper bound on the real edge coverage.

For small graphs, it is possible to look for a global optimum with our approach by exploring non deterministically all admissible decompositions. In practice, we have limited our study to incremental, locally optimal searches in order to manage the large graphs that occur in bioinformatics. An intermediate approach is possible. At each step, the local optimum is not necessarily unique. If two optimal concepts are compatible, they can be chosen in a same step. If they are not compatible, it introduces a choice point that can be subject to backtracking. In fact, even in this case we need to better manage the dependencies in the graph contexts in relation with the constraints to be checked in order to reduce the complexity of the search. The next proposition is a first step in this direction.

2.3 Exploitation of the Graph Context to Reduce the Search Space

If the number of nodes of a graph is reduced by a factor k, its graph context is reduced by a factor k^2 and the search space of concepts is reduced by a factor 2^{k^2} . This emphasizes the well known importance of reduction strategies as a preprocessing step of a concept search. Simple techniques include the application of standard clarification and reduction procedures and the decomposition of the

graph into connected components. Concerning the FCA reduction procedures and since we are interested in largest concepts, the computation of the edge cover size requires all reduced nodes to have an associated weight counting the number of nodes they represent. Concerning the decomposition of the graph, we propose a generalization of the connected component property aiming at further splitting the graph context. This "bipartite split" property can be applied recursively, in order to work on increasingly smaller contexts, but the gain is generally weak.

Dot operator: The dot operator on sets of objects or attributes is introduced in order to ease the bipartite split expression. It is a relaxed variant of the derivation operator of FCA where the universal quantification is replaced by an existential one. Given a set of objects X (resp. attributes Y), the set $\dot{X}$ (resp. $\dot{Y}$) is made of all attributes (resp. objects) related to *at least one* attribute in X (resp. Y):

$$\dot{X} = \{y \in Y \mid \exists x \ \in \ X,\ r(x,y)\} \qquad \dot{Y} = \{x \in X \mid \exists y \ \in \ Y,\ r(x,y)\} \tag{1}$$

As for derivation, the dot operator can be combined multiple times:

$$\ddot{X} = \{x \in X \mid \exists y \in \dot{X},\ r(x,y)\} \qquad \ddot{Y} = \{y \in Y \mid \exists x \in \dot{Y},\ r(x,y)\} \tag{2}$$

Proposition: Given a set of objects O and a set of attributes A, let $P = \{O_1, O_2\}$ be a partition of O and $Q = \{A_1, A_2\}$ be a partition of A. Let $LC(O, A)$ denotes a largest concept of the formal context $\mathcal{C}(O, A)$, i.e. a concept corresponding to a submatrix of largest size. Then, the following property holds:

$$LC(O, A) = \max\left(LC(O_1, A_1), LC(\dot{A}_2 \cup \ddot{O}_2, \dot{O}_2 \cup \ddot{A}_2)\right) \tag{3}$$

Moreover, this equation may be refined if no relation holds over $O_1 \times A_1$:

$$LC(O, A) = \max\left(LC(\dot{A}_2, A_2), LC(O_2, \dot{O}_2)\right) \tag{4}$$

Proof: Let $P = \{O_1, O_2\}$ be a partition of a set of objects O and $Q = \{A_1, A_2\}$ be a partition of a set of attributes A. If the largest concept $LC(O, A)$ is in the context $\mathcal{C}(O_1, A_1)$, then by definition, it will take the right value $LC(O_1, A_1)$. Else, there exists an element of the largest concept either in O_2 or in A_2 . If the element is in O_2, then the attributes of $LC(O, A)$ have to be included in $\dot{O}_2$. The same way, the objects of $LC(O, A)$ have to be included in the objects sharing at least one relation with attributes of $\dot{O}_2$, that is, $\ddot{O}_2$. With a symmetric argument, if there is an element of A_2 in the largest concept, then attributes of $LC(O, A)$ have to be included in $\dot{A}_2$ and objects in $\ddot{A}_2$. Altogether $LC(O, A)$ must be a concept of the context $\mathcal{C}(\dot{A}_2 \cup \ddot{O}_2, \dot{O}_2 \cup \ddot{A}_2)$.

If no relation holds over $O_1 \times A_1$, then every concept has either all its elements in O_2 or all its elements in A_2. In the first case they are in the formal context $\mathcal{C}(O_2, \dot{O}_2)$, and in the second case they are in the formal context $\mathcal{C}(\dot{A}_2, A_2)$. The largest concept is the largest of the largest concepts of the two contexts. Table 2 gives details on the way the search can be split in this case. □

From the point of view of graph modeling, the fact that no relation holds over $O_1 \times A_1$ means that it is stable set of the graph (note however that O_1 and

		A_1			A_2			
		H	I	J	K	L	M	N
O_1	a					x	x	
	b							
	c				x	x	x	x
O_2	d			x		x	x	x
	e	x		x	x			
	f	x		x	x		x	x
	g			x	x		x	x

Table 2: A partitioned context with no relation over $\mathcal{C}(O_1, A_1)$. The five possible positions of the largest concept $\mathcal{L}$ are shown. $\mathcal{L}$ could be in $\mathcal{C}(O_2, A_1)$ (e.g. $(\{e, f\}, \{H\})$), $\mathcal{C}(O_2, A_2)$ (e.g. $(\{f, g\}, \{M, N\})$), $\mathcal{C}(O_1, A_2)$ (e.g. $(\{a\}, \{L, M\})$), $\mathcal{C}(O_2, A_1 \cup A_2)$ (with an element in A_1 and an element in A_2, e.g. $(\{e, f, g\}, \{J, K\})$) or $\mathcal{C}(O_1 \cup O_2, A_2)$ (e.g. $(\{c, d\}, \{L, M, N\})$).

A_1 may overlap). If moreover no relation holds over $O_2 \times A_2$, it corresponds to the existence of at least two connected components in the graph. If $O_1 = A_1$, O_1 is a stable set and it can be searched by looking for cliques in the graph's complement. Moreover, we get $O_2 = A_2$, so $LC(\dot{A}_2, A_2) = LC(O_2, \dot{O}_2)$ and it is sufficient to consider the graph context $\mathcal{C}(O, \dot{O}_2)$. Since the aim is to speedup the search for concepts, the search in bounded time of a "best" clique (not necessarily maximal) has to be achieved, a feature allowed by ASP.

3 *PowerGrASP*, graph compression based on FCA

We propose an implementation of *Power graph* compression[1] as a formal concept search using a non-monotonic logical formalism, ASP, to model the constraints on the power nodes and power edges introduced in section 1.2.

ASP (Answer Set Programming) is a form of purely declarative programming oriented towards the resolution of combinatorial problems [20]. It has been successfully used for knowledge representation, problem solving, automated reasoning, and search and optimization. In the sequel, we rely on the input language of the ASP system Potassco (*Potsdam Answer Set Solving Collection* [13] developed in Potsdam University. An ASP program consists of Prolog-like rules h :- b_1, ..., b_m, *not* b_{m+1}, ..., *not* b_n, where each b_i and h are literals and *not* stands for *default negation*. Mainly, each literal is a predicate whose arguments can be constant atoms or variables over a finite domain. Constants start with a lowercase letter, variables start with an uppercase letter or an underscore (don't care variables). The rule states that the head h is proved to be true (h is in an answer set) if the body of the rule is satisfied, i.e. $b_1, \ldots, b_m$ are true and one can not prove that $b_{m+1}, \ldots, b_n$ are true. Note that the result is independent on the ordering of rules or of the ordering of literals in their body. An ASP solver can compute one, several, or all the answer sets that are solutions of the encoded problem. If the body is empty, h is a fact while an empty head specifies

[1] Repository at `http://powergrasp.bourneuf.net`

an integrity constraint. Together with model minimality, interpreting the program rules this way provides the stable model semantics (see [15] for details). In practice, several syntactical extensions to the language are available and we will use two of them in this paper: the choice rules and optimization statements.

A *choice rule* of the form $i\{h : b_1, \ldots, b_n\}j$, where i and j are integers, states that at least i and at most j grounded h are true among those such that the body is satisfied. In the body part, $N = \{h\}$ evaluates N to the cardinal of the set of h. An optimization statement is of the form $\#operator\{K@P, X : b_1, \ldots, b_n\}$, where *operator* is the type of optimization (either maximize or minimize), K refers to integer weights whose sum as to be optimized, P is an optional priority level in case of multiple optimization (the highest is optimized first) and X is a tuple of variables such that one weight K is associated to each tuple of values of these variables. The Potassco system [14] proposes an efficient implementation of a rich yet simple declarative modeling language relying on principles that led to fast SAT solvers. ASP processing implies two steps, grounding and solving. The grounder generates a propositional program replacing variables by their possible values. The solver is in charge of producing the stable models (answer sets) of the propositional program.

The main objective of this section is to show, through its application to the stated graph compression problem, the effectiveness and flexibility of ASP for modeling various searches in the concept lattice. Of course, it remains always preferable from the point of view of efficiency to design a specific algorithm for a particular FCA problem. However, ASP systems are not "toy" environments and are useful for the design of efficient prototypes or to take into account a knowledge-rich environment while keeping a simple code. ASP has shown to be an attractive alternative to standard imperative languages that enable fast developments and is most of the time sufficient in real applications where many constraints have to be managed. In fact, Potassco proposes an integration of ASP and Python that allows the development of hybrid codes. The reminder of this section provides the relevant part of the encoding for solving variants of *Power graph* compression. All lines starting with % are comments.

The initial graph is assumed to be coded with facts $edge(X, Y)$, where X and Y are vertices (e.g. $edge(a, d)$. $edge(a, e)$.) If the graph is non oriented and not bipartite, edges are made symmetric by adding a clause $edge(X, Y) : -edge(Y, X)$. Many input graph examples are available in the *PowerGrASP* source code[2].

3.1 Looking for a formal concept and for graph motifs

Listing 1.1 provides a code specifying a formal concept. They are represented by a predicate of arity 2: $concept(S, X)$ holds if the node X is in the set number S. For instance, the concept $(\{a\}, \{b, c\})$ is represented by $concept(1, a)$, $concept(2, b)$ and $concept(2, c)$. The elements of a concept (at least 1 in each set) are chosen among those linked by an edge (line 2-3). The set of admissible concepts is restricted by three constrains: the necessity to have two disjoint sets,

[2] `powergrasp/tests/`

```
1 % Choice of elements in set 1 and 2 of a concept
2 1 { concept(1,X): edge(X,_)}.
3 1 { concept(2,Y): edge(_,Y)}.
4 % Bipartite subgraph: The two sets are disjoint.
5 :- concept(1,X), concept(2,X).
6 % A node is impossible in a set if not linked to some node in the other set
7 imp(1,X):- edge(X,_), concept(2,Y), not edge(X,Y), X!=Y.
8 imp(2,Y):- edge(_,Y), concept(1,X), not edge(X,Y), X!=Y.
9 % Consistency ; no impossible element can be added to the concept
10 :- imp(T,X), concept(T,X).
11 % Maximality ; all possible elements have to be added to the concept
12 :- not imp(1,X), not concept(1,X), edge(X,_).
13 :- not imp(2,Y), not concept(2,Y), edge(_,Y).
```

Listing 1.1: Search of concepts: maximal bicliques

```
1 % Choice of elements in the clique
2 3 { clique(X): edge(X,_)}.
3 % A node is impossible in a set if not linked to some node in the other set
4 imp(X):- edge(X,_), clique(Y), not edge(X,Y), X!=Y.
5 imp(Y):- edge(_,Y), clique(X), not edge(X,Y), X!=Y.
6 % Consistency ; no impossible element can be added to the clique
7 :- imp(X), clique(X).
8 % Maximality ; all possible elements have to be added to the clique
9 :- not imp(X), not clique(X), edge(X,_).
10 :- not imp(Y), not clique(Y), edge(_,Y).
```

Listing 1.2: Search of concepts: maximal cliques

the necessity to have a biclique between these sets (line 10) and the necessity to have a maximal biclique(line 12-13). The two last constraints make use of a common symmetric predicate *imp*() that lists elements that cannot be in a set if another element is chosen in the other set (line 7-8).

Powergraphs contain two types of motifs, bicliques and cliques. The first one is searched by looking for all solutions of the previous code. Cliques can be considered as a simplification where the two sets are equal and deserves a special code (listing 1.2). A clique must contain at least 3 elements. The loops on each elements are not required since in most applications they are not present.

The code that ensures generated concepts do not break the power node hierarchy is not presented here for the sake of brevity (see the *PowerGrASP* source code[3]). It mainly consists in the maintenance of an inclusion tree of power nodes. The power edge decomposition condition is met by design : once a concept is chosen, its edges are removed from the compressed graph and cannot therefore be involved in the remaining concepts.

[3] `powergrasp/ASPSources/postprocessing.lp`

```
1 % The concept score is its edge cover.
2 score(N1*N2):- N1={concept(1,X)}, N2={concept(2,X)}.
3 % Exclusion of concepts with unbounded score.
4 :- score(S) ; S>upperbound.
5 :- score(S) ; S<lowerbound.
6 % Maximize the score.
7 #maximize{S@1,S:score(S)}.
```

Listing 1.3: Concept ordering

3.2 Implementing Concept Scoring

Listing 1.3 implements the scoring of concepts. It simply maximizes the edge cover of found concepts (line 7), which is computed as the product of the number of nodes in each set (line 2) and contained between two constant parameters of the program, *lowerbound* and *upperbound*. By default the lowerbound equals 2 (edge coverage of a star involving three nodes) and the upperbound is the minimum between the size of the previous chosen concept and the number of remaining edges in the connected component.

Note that in the real code, maximization is replaced by a minimization of `uppperbound - score`. This leads to a significant speedup because the solver converts maximization to minimization and uses higher upperbound values. Likewise, the code in Listing 1.3 is able to handle directed graph but a dedicated slightly more efficient version exists for this type of graph[4].

3.3 Implementing the Stable Search and Heuristics

Stables are used to split the search space as exposed in *proposition 1* (see section 2.3). They are searched in the complement context, i.e. with 0 and 1 inverted, using the code defined in listing 1.1 without the bipartite constraint (line 5).

The implementation of heuristics takes only a few ASP lines. The code of HDF is provided in listing 1.4 (*PowerGrASP* [5]) predicate *interest_ok* states that one of the concept nodes must be chosen among top interesting nodes. It uses *max_interest*, a predicate true only for nodes with highest interest in the current graph. In the *PowerGrASP* source, the computation of this atom is performed on the Python side. The code of FHDF is provided in Listing 1.5 (*PowerGrASP* [6]). It needs each node to get a degree of interest computed in Python and coded in predicate *interest_level*. Line 2 maximizes the sum of node degrees belonging to the concept. Note that the priority of the interest maximization is greater than the priority of the score maximization, so that the heuristics is used as a first selection criterion. For the K2HDF heuristics, the code remains the same than in listing 1.4 but the Python part is relaxing the constraint by generating *max_interest* for the nodes of maximal degree plus the nodes sharing a maximal number of common neighbors with another node.

[4] `powergrasp/ASPSources/findbestorientedbiclique.lp`
[5] `powergrasp/ASPSources/by_priority.lp`
[6] `powergrasp/ASPSources/by_fuzzy_priority.lp`

```
% At least one node of maximal priority is involved in the concept.
interest_ok :- concept(_,X) ; max_interest(X).
:- not interest_ok.
```

Listing 1.4: Addon to the (bi)clique search program: HDF and K2HDF heuristics.

```
% Maximize the concept priority as a sum of node degrees of set 1.
#maximize{P@2,X: concept(_,X), interest_level(X,P)}.
```

Listing 1.5: Addon to the (bi)clique search program: FHDF heuristics.

4 Biological benchmarks

The powergraph compression code is benchmarked here using three networks coming from biological data. Our aim is just to show that our ASP code is already useful in managing real graphs. A complete evaluation on a larger benchmark is out of the scope of this paper, which mainly tries to take the first steps in FCA applied to the powergraph issue.

The network named `rna` comes from a study on the pea aphid (*A. pisum*) [27]. It is a bipartite interaction graph linking two disjoint populations of molecules (15×1810). The second network, `mdb`, comes from the database MatrixDB [19] describing interactions between extracellular proteins. It contains 5 connected components, the largest one involving 273 nodes and 642 edges. The third graph, `sbind` comes from biological data of Royer *et al.*, and contains one large connected component (205 nodes, 335 edges) and 188 with less than 20 nodes. These three graphs are described in Table 3, and the compression results in Table 4. A graphical representation of the `rna` network is shown in Figure 2. All benchmarks have been run on one core of an *Intel i7-6600U* (2.60GHz).

Bipartite stable search. As shown in section 2.3, the bipartite stable is the object needed by *proposition 1* in order to restrict the search space. It is computed with a time limit of 5 seconds ; its optimality is thus not guaranteed, but not required either. Results show that stables of consequent size are found (see Table 3). In case of the `rna` bipartite network, 9 out of 16 nodes of the first set and one third of the second are involved in the stable. This result suggests that finding a large stable is an interesting application of *proposition 1.*

	#node	#edge	density	#cc	bipartite	stable
rna	1825	2250	0.0013	1	yes	9x765
mdb	286	652	0.016	5	no	171x213
sbind	864	1241	0.003	189	no	102x122

Table 3: Network statistics. *#node*: number of nodes in the graph; *#edge*: number of edges; *density*: global density of the graph; *#cc*: number of connected components in the graph; *stable*: size of the stable found in the largest connected component.

	regular			HDF			FHDF			K2HDF		
	rna	mdb	sbind	rna	mdb	sbind	rna	mdb	sbind	rna	mdb	sbind
time (s)	380	80	50	**177**	**25**	**25**	5460	150	40	320	40	**25**
edge reduction	**96%**	**64%**	**61%**	94%	47%	50%	77%	50%	53%	95%	54%	52%
#powernode	48	83	206	55	41	231	409	103	244	64	76	233
#poweredge	49	96	182	55	45	200	430	144	229	64	66	193
#remain	50	133	301	81	300	414	91	182	356	57	226	406
constraint ratio	1293.3	124	49.2	**32.0**	124	**37.5**	1293.3	124	**37.5**	1293.3	124	**37.5**
conflict ratio	1.49	2.32	0.2	**0.12**	2.18	**0.11**	**0.13**	**1.1**	0.27	**0.1**	2.81	0.16

Table 4: Network compression results. *Compression time*: time needed to fully compress the graph (iterative search of all concepts); *edge reduction*: given by $\frac{\#initial\ edges - \#poweredges}{\#initial\ edges}$; *#powernode*: number of power nodes in the compressed graph; *#poweredge*: number of power edges; *constraint ratio*: the ratio between constraint numbers for the first biclique search, and edge cardinality; *conflict ratio*: the ratio between conflict numbers for the first biclique search, and edge cardinality.

	Time (s)		Edge reduction	
	Royer *et al.*	*PowerGrASP*	Royer *et al.*	*PowerGrASP*
rna	4	380	96%	96%
mdb	2	80	64%	64%
sbind	1	50	61%	61%

Table 5: Comparison of *PowerGrASP* and Royer *et al.* implementation.

Comparison with the Royer et al. *implementation* Oog As shown in Table 5, *PowerGrASP* reaches a score equivalent to *Oog*. In fact, since both implement the same strategy, edge reduction is very similar up to slight variations due to non-determinism of motif choices with equal scores. *Oog* uses a dedicated algorithm, making it much more scalable. Our goal at this step was to produce a much more flexible framework however so that it could lead to qualitatively better results with different strategies and the inclusion of background knowledge.

Benchmarks of the four methods. Benchmarks show that *regular* compression (Royer *et al.* heuristic) yields best edge reduction in all cases, but is generally slower. The other heuristics are faster, but reach a smaller edge reduction. *HDF* is the fastest, which is expected because of (1) the easy computation of a node degree of interest and (2) the important restriction of search space it induces, as shown by the number of constraints and conflicts generated (the number of constraints is related to the size of the problem to be solved, conflicts express the practical combinatorial complexity of the search). *HDF* yields the smaller edge reduction. *FHDF* and *K2HDF* bring slight improvements of edge reduction over *HDF*, at the cost of more compression time. The time needed to compress `rna` graph with *FHDF* is explained by the computation cost that increases with the number of nodes (1825 for `rna`). *K2HDF* reaches a slightly higher edge reduction mostly because the constraint is relaxed, allowing the exploration of a slightly larger subset of concepts.

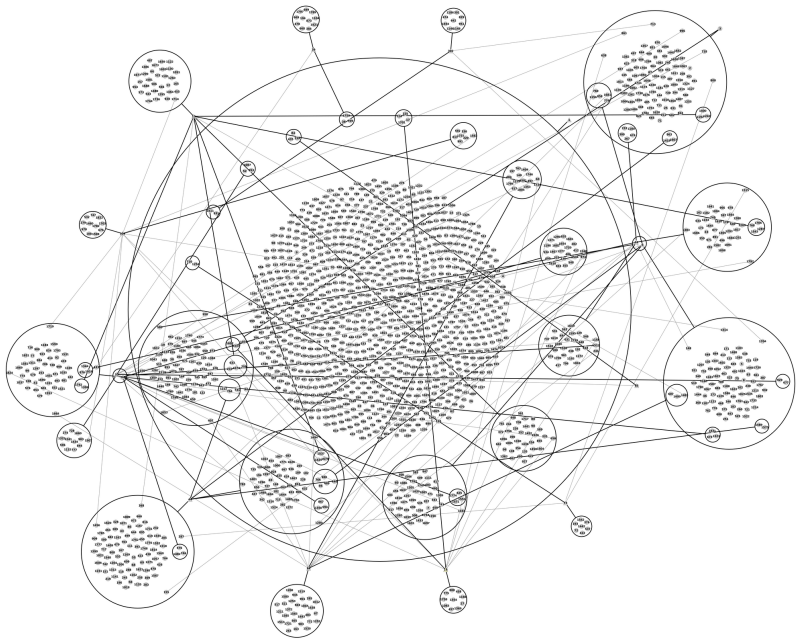

Fig. 2: Cytoscape view of a bipartite RNA network compressed with *Power-GrASP* where two types of molecules interact. The 15 nodes with a triangle shape belongs to the so-called *microRNA* type and appear around the central power node that contains most of the molecules of the *mRNA* type.

5 Discussion and Conclusion

The main goal of this paper was to introduce the *Power graph* compression issue, a knowledge-discovery oriented kind of graph compression, in terms of a decomposition problem in the formal concept analysis framework. The Galois concept lattice is the fundamental structure underlying the search space of this problem. If Power Graph compression is reducible to a choice (and an ordering) of concepts, one can thus use FCA to formalize (1) algorithms of motifs enumeration (HDF for instance), (2) reduction methods of the search space (prop. 1 for instance), and (3) other approaches using FCA extensions like TCA and pattern structure to obtains different visualizations.

For real applications, it is reasonable to look for an approximation of optimal decompositions. We have proposed both concept generation heuristics and a splitting strategy to reduce this search space. From the point of view of implementation, we have shown that a high level purely declarative approach using Answer Set Programming allows a compact and flexible encoding of combinatorial searches in the space of concepts. Moreover, the resulting program is efficient enough to handle formal contexts of reasonable size. Large biological networks can be compressed in few minutes as shown in section 4, allowing a hierarchical clustering of many biological entities based on network topology.

Optimal compression of graphs including thousands of nodes is the ultimate goal of this research, a goal that is not reached by the Royer *et al.* heuristic, nor by the variants described in section 3. Our current implementation does not works efficiently on large dense graphs. This is partly due to ASP memory consumption since grounding needs all data to be encoded and loaded in memory in order to explore the search space. Thus reduction strategies are of uttermost importance to split the search space.

A number of improvements can be studied in future work along two main research tracks. From a theoretical perspective, the links between an optimal covering and an optimal decomposition in bicliques and cliques should be considered. It could be interesting to start from a solution for the covering problem, allowing some edges to be covered several times and then refine this solution to comply with all the constraints. This approach would be more compatible with the FCA framework and a search for a global optimum. Moreover, some overlapping configurations may be easily understandable in terms of application and it may be interesting to relax some of the powergraph conditions. From an application perspective, one could search for quasi-motifs, that is, motifs with a few missing edges, which are known to groups proteins of the same family in proteomes [5], and improve the resistance against noisy an incomplete data. This method was used to treat various real-life graphs to reveals important patterns [2]. Finally, knowledge discovery requires to integrate domain specific knowledge like annotations, and metadata on the quality or significance of data.

Acknowledgments We wish to thank D. Tagu (INRA Le Rheu) and N. Théret (Inserm) for providing us the networks used in the results section. We would also like to express our gratitude to the reviewers for their feedbacks.

References

1. P. K. Agarwal, N. Alon, B. Aronov, and S. Suri. Can visibility graphs be represented compactly? *Discrete & Computational Geometry*, 12(3):347–365, 1994.
2. S. E. Ahnert. Generalised power graph compression reveals dominant relationship patterns in complex networks. *Scientific reports*, 4, 2014.
3. G. Alexe, S. Alexe, Y. Crama, S. Foldes, P. L. Hammer, and B. Simeone. Consensus algorithms for the generation of all maximal bicliques. *Discrete Applied Mathematics*, 145(1):11 – 21, 2004. Graph Optimization {IV}.
4. J. Amilhastre, M. Vilarem, and P. Janssen. Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs. *Discrete Applied Mathematics*, 86(2–3):125–144, 1998.
5. D. Bu, Y. Zhao, L. Cai, H. Xue, X. Zhu, H. Lu, J. Zhang, S. Sun, L. Ling, N. Zhang, G. Li, and R. Chen. Topological structure analysis of the protein–protein interaction network in budding yeast. *Nucleic Acids Research*, 31(9):2443–2450, 2003.
6. M. Chein. Algorithme de recherche des sous-matrices premiÈres d'une matrice. *Bulletin mathématique de la Société des Sciences Mathématiques de la République Socialiste de Roumanie*, 13 (61)(1):21–25, 1969.
7. F. Chung. On the coverings of graphs. *Discrete Mathematics*, 30(2):89 – 93, 1980.

8. S. Daminelli, V. J. Haupt, M. Reimann, and M. Schroeder. Drug repositioning through incomplete bi-cliques in an integrated drug–target–disease network. *Integrative Biology*, 4(7):778–788, June 2012.
9. T. Dwyer, N. Henry Riche, K. Marriott, and C. Mears. Edge compression techniques for visualization of dense directed graphs. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2596–2605, Dec. 2013.
10. T. Dwyer, C. Mears, K. Morgan, T. Niven, K. Marriott, and M. Wallace. Improved optimal and approximate power graph compression for clearer visualisation of dense graphs. *CoRR*, abs/1311.6996, 2013.
11. D. Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information Processing Letters*, 51(4):207 – 211, 1994.
12. J. Gagneur, R. Krause, T. Bouwmeester, and G. Casari. Modular decomposition of protein-protein interaction networks. *Genome Biology*, 5(8):R57, July 2004.
13. M. Gebser, R. Kaminski, B. Kaufmann, , M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, and S. Thiele. Potassco user guide. 2015.
14. M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124, 2011.
15. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *Proc. of the 7th International Conf. on Logic Programming (ICLP)*, pages 579–97, 1990.
16. P. Hitzler and M. Krötzsch. Querying formal contexts with answer set programs. In *Proc. of the 14th Int. Conf. on Conceptual Structures: Inspiration and Application*, ICCS'06, pages 260–273. Springer-Verlag, 2006.
17. S. Jukna and A. Kulikov. On covering graphs by complete bipartite subgraphs. *Discrete Mathematics*, 309(10):3399 – 3403, 2009.
18. A. D. King, N. Pržulj, and I. Jurisica. Protein complex prediction via cost-based clustering. *Bioinformatics*, 20(17):3013–3020, Nov. 2004.
19. G. Launay, R. Salza, D. Multedo, N. Thierry-Mieg, and S. Ricard-Blum. Matrixdb, the extracellular matrix interaction database: updated content, a new navigator and expanded functionalities. *Nucleic Acids Research*, 43(D1):D321–D327, 2015.
20. V. Lifschitz. What is answer set programming? In *Proc. of the 23rd National Conf. on Artificial Intelligence - Vol. 3*, AAAI'08, pages 1594–97. AAAI Press, 2008.
21. S. Navlakha, M. C. Schatz, and C. Kingsford. Revealing biological modules via graph summarization. *Journal of Computational Biology*, 16(2):253–264, 2009.
22. H. Ogata, W. Fujibuchi, S. Goto, and M. Kanehisa. A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. *Nucleic Acids Research*, 28(20):4021–4028, Oct. 2000.
23. L. Royer, M. Reimann, B. Andreopoulos, and M. Schroeder. Unraveling Protein Networks with Power Graph Analysis. *PLoS Comput Biol*, 4(7):e1000108, 2008.
24. S. Rudolph, C. Săcărea, and D. Troancă. Membership constraints in formal concept analysis. In *Proc.of the 24th Int. Conf. on Artificial Intelligence*, IJCAI'15, pages 3186–3192. AAAI Press, 2015.
25. P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, and D. e. a. Ramage. Cytoscape: A software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13(11):2498–2504, 2003.
26. G. Tsatsaronis, M. Reimann, I. Varlamis, O. Gkorgkas, and K. Nørvåg. Efficient community detection using power graph analysis. In *Proc. of the 9th Workshop on Large-scale and Distributed Informational Retrieval*, pages 21–26. ACM, 2011.
27. V. Wucher. *Modeling of a gene network between mRNAs and miRNAs to predict gene functions involved in phenotypic plasticity in the pea aphid.* Thesis, Université Rennes 1, Nov. 2014.